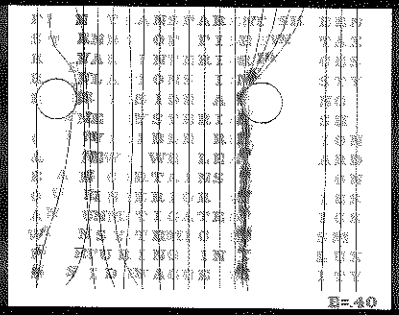
Roberto Simanowski
Jörgen Schäfer
Peter Gendolla (eds.)

Reading Moving Letters



Digital Literature in Research and Teaching

A Handbook

[transcript] Media Upheavals 40

Snow, C. Percy. *The Two Cultures and the Scientific Revolution.* Cambridge, MA: Cambridge UP, 1964.

Sunstein, Cass. R. *Republic.com 2.0.* Oxford: Princeton UP, 2007.

Thomas, Sue et al. "Transliteracy: Crossing Divides." *First Monday* 12.12 (2007). 20 July 2009 <http://www.uic.edu/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/2060/1908>.

Wardrip-Fruin, Noah. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies.* Cambridge, MA: MIT P, 2009.

Noah Wardrip-Fruin

Learning to Read Digital Literature

1 Why Read Digital Literature?

Despite the length of my essay for this volume on the terminology and concepts I use in my thinking about digital literature, a major question remains unaddressed: Why read digital literature?

We might initially answer: For all the same reasons we read other literature. And given that writers work to place their creations wherever there are readers, we can be assured of an increasing literary presence on all the computer-driven screens where we read and write—our laptops, desktops, handhelds, cellphones, console-connected televisions, and so on.

But I believe there are a number of other important reasons at this moment for discussing what we might mean by "reading" digital literature—and learning to perform such readings.

First, as Ted Nelson began arguing in the 1970s, we're living in a world increasingly defined by computer systems—systems designed and implemented by humans. These systems can be designed poorly, or implemented poorly, or designed and implemented to help some people and make life difficult for others . . . but this is the fault of humans, and it can be corrected (and sooner rather than later, if we can learn to spot bad designs before they're widely adopted). To put it another way, "the computer just works that way" is a non-argument. The importance of this knowledge lay behind Nelson's now-famous cry from the front cover of *Computer Lib/Dream Machines*: "You can and must understand computers NOW." When we study works of digital literature—and digital art more generally—in the manner I advocate, we study computational systems explicitly as authored artifacts crafted toward particular ends and embedded in particular contexts, rather than as black boxes that produce neutral outputs "the way the computer works." I believe this represents a crucial form of literacy for our current culture.

Second, in a more specific instance of the above, we're entering a period in which the results of computational processes are increasingly used to form assumptions or offered as evidence. This is one thing if we're forming our assumptions about whether the weekend will be sunny while we're trying to decide whether to have a picnic—but the results of computer simulations are also increasingly used when we're in the process of trying to make more weighty decisions about matters such as city planning and greenhouse gas

emissions. To take one of my favorite examples, Jay Forrester's urban dynamics simulations (which inspired *SimCity*) can be used to try to figure out how to build a healthy city, but we need to view any results from Forrester's work through an interpretation of the structures and processes of the simulations—which Garn and others have argued are deeply flawed (for example, by their cities' lack of dynamic interaction with suburbs). We need to learn to ask questions of the designs of simulations that are analogous to the questions we ask when presented with other forms of evidence (e.g., Was the study double blind? What's the n?). Unfortunately, because these questions will often have to do with the unexamined assumptions of the simulation's designers, it may only be possible to pose such questions after literate and informed examination of the simulation's processes. Here, again, learning to read digital literature in a well-rounded manner will help us develop a critical practice that is not limited to inquiry along the lines considered relevant by the designers of such simulations.

Third, and this is particularly important to me as a writer and artist, we should learn to read digital literature because we're increasingly using computational systems as a means of expression and (just as careful reading of exemplary literature is central to the development of most writers) careful reading of digital literature systems is important to our development. Those of us with some computer science coursework under our belts have learned something about reading such systems in terms of things like computability and efficiency. And those of us who've spent some time around computing subcultures have probably learned something about reading them in terms of formal elegance (cf. Maurice Black's *The Art of Code*). But we're only beginning to learn to read computational systems in terms of what they express.

2 Procedural Literacy

In the context of the arts and humanities, most of what I suggest that we study in my essay for this volume is quite familiar. While "data" may be an unusual term for them, we're certainly accustomed to studying text, images, sounds, and so on. Our practices for studying social context, performance, and most other topics I've discussed are well-developed, and already work is taking place that employs them in interpreting digital literature and art.

But there is one aspect that is glaringly absent from our curricula: the study of computational processes. And, as hinted above, we can't address this simply by sending our students to computer science classes. While computer science may teach students about data structures and algorithms, the forms of interpretation explored in such courses are normally limited to issues such as

efficiency, maintainability, and elegance. We want our students to have a grasp of algorithms, of processes, but through a very different interpretive lens (and through a set of examples with a different emphasis than those used in most introductions to computer science).

One way to address this would be to make common cause with the group, largely within computer science and digital media programs, that is calling for "procedural literacy." This call is one that, for being associated with computers, has quite a long history. In fact, it follows a strand back into the history of computer science education that can be seen in the 1960s, for example in the observations of Alan Perlis—the first winner of the Turing award (computer science's greatest honor). He proposed that every student (in every discipline) take a first course in programming as part of a general education. Particularly interestingly for our purposes, in 1961 he said:

> Perhaps I may have been misunderstood as to the purpose of my proposed first course in programming. It is not to teach people how to program a specific computer, nor is it to teach some new languages. The purpose of a course in programming is to teach people how to construct and analyze processes. I know of no course that the student gets in his first year in a university that has this as its sole purpose. (Greenberger 206)

Perlis's argument here is cited in several of the writings of pioneering computer science educator Mark Guzdial, who has been working to design first courses in programming that would be appropriate for such use (Guzdial). His results are so far quite encouraging—while the traditional introduction to computer science courses at Georgia Tech have (like those at most major research universities) a high attrition rate, especially among non-majors, his courses have (a) retained students well and (b) succeeded in teaching them about constructing and analyzing processes. However, the types of examples presented and the types of interpretation involved are not ideal for students of digital literature—that is, for developing the ability to perform the sorts of readings advocated in my essay in part one of this volume.

On the other hand, Michael Mateas (while Guzdial's colleague at Georgia Tech) designed a course aimed specifically at digital media practitioners and theorists, with the goal of procedural literacy: *Computation as an Expressive Medium*. Mateas writes:

> By procedural literacy I mean the ability to read and write processes, to engage procedural representation and aesthetics, to understand the interplay between the culturally-embedded practices of human meaning-making and technically-mediated processes. (Mateas 101)

Having read Mateas's syllabus, met some of his students from this course, seen their projects, and talked with Mateas about his overall approach and goals, I believe his course is exceptionally well-designed. (On a technological level, Mateas's course begins by using Processing—a set of abstractions on top of Java designed by artist/programmers Ben Fry and Casey Reas—and then moves on to raw Java). I also think the goal of procedural literacy that he articulates would be especially valuable for those learning to read digital literature, for the following reasons:

- It would be valuable for students to develop an understanding of processes as designed artifacts. Understanding common structures of computational processes in a general sense would open digital media systems to informed modes of interpretation along these lines, even when the source code for the project is not available. Online, Mateas has referred to Kurt Squire's procedurally-focused reading of *Viewtiful Joe* (which I assume is that found in "Educating the Fighter") as a successful example of this.

- Students would emerge being able to communicate more effectively with those who design processes for digital media systems. This would make it more possible to interview artists who use such processes in their work and to engage developer communities more generally. It would also make collaboration with computer scientists and programmers much easier—whether with the goal of creating works of digital literature or of fashioning tools to aid in their analysis (or aid in the communication of results).

- Finally, of course, it would be useful for just about anyone to have practice designing processes, and to have learned some of the helpful concepts and practices such a course could include. Even executing a complex series of find-and-replace operations in a word processor is often easier for those who have had some training in designing computational processes at a general level.

But procedural literacy is not enough.

3 Beyond Procedural Literacy

3.1 Those Despised Specifics

From Perlis's 1961 words to Mateas's more than forty years later, there's a certain disdain for specifics—of particular languages, of particular development environments. As Mateas puts it, "it is not the details of any particular programming language that matters, but rather the more general tropes and structures that cut across all languages" (Mateas 101). Certainly this is true to some extent, but there are also important parts of digital culture, ones quite relevant to those interested in digital literature, that we simply can't access without some understanding of such specifics.

In part this is because some of the most creative computational subcultures (e.g., the demoscene) are focused specifically on work that is interesting and/or difficult precisely in the ways in which it is shaped by the particularities of given computer hardware or software. But even outside such subcultures, many digital media practitioners are actively engaged in reflecting on or working against the specifics of their work environments. The interactive fiction (IF) community, for example, is not organized around difficult feats of programming (the way the demoscene is) but it still is home to practices that reflect engagement with the specifics that Perlis and Mateas would bracket. For example, there is a practice of creating "one room" works of IF, which derives part of its interest from the fact that—for a number of years, ending with the release of Inform 7—all the common IF development environments were object-oriented systems designed around supporting multiple rooms with connections to one another.[1] Authors giving up this fundamental aspect of these tools is difficult to interpret if we're unprepared to consider the specifics of tools. And we'd be at even more of a loss to explain what is virtuosic (and amusing) about creating a work of IF, available for the Z-machine, that includes a chess-playing program.[2] In other words, we need to be able to engage work that is interesting, perhaps virtuosic, because of how it interacts with technical specifics. An introduction for those who would study and make digital literature shouldn't just be about "procedural literacy"—it should also include some engagement with the specifics of particular environments, so that students will learn something about this kind of engagement.

Which brings us to what I consider the good news: there's absolutely no way to achieve the dream of an introduction to procedural literacy that brackets all specifics. For example, as pointed out above, Mateas's course engages the specifics of Processing and then Java. My point is that the engagement with such specifics shouldn't be bemoaned. It's one of the important parts of

what students are learning. And we should choose the particular development environments employed, in part, for what they teach about specifics and for the health of their digital media development communities—rather than, for example, some near-evangelical hatred of languages with too much "syntactic sugar." Mateas's choice, of Processing and Java, is a good one in this regard. But there are also others that it would make sense to pursue. For example, now that Actionscript provides a more full language than present in its first version, as well as the ability for code to be written in an external editor, it might make sense to teach a course such as Mateas's using Flash (if working with students in a subfield for which it is a primary tool). In computer music circles, Max/MSP and Jitter—or Pd (Pure Data)—might be the most appropriate choice.

3.2 What about CS?

Given that computer science, as a discipline, isn't very interested in the issues that are primary to practitioners and scholars of digital literature (and digital media generally) it should be no surprise that most who speak of procedural literacy bracket CS almost entirely. And this outlook is supported by the anecdotal evidence of students interested in digital media who take introductory computer science courses—they learn three data structures and four sorting algorithms, briefly wonder what relationship any of it could have to what interests them, and then promptly forget it. But I would argue that we can't afford to completely bracket computer science, because:

- Digital media is built on computer science research results. Students who don't know the basic CS vocabulary in areas related to digital media, and who have no practice reading CS research papers, can't even do a full review of the literature for their field.
- This research is often ongoing. Students who know something of the vocabulary and conceptual framework of CS as it relates to digital media will have a much broader field of examples to consider, potential collaborators, and technologies at their disposal. Those who don't know this are in danger of reinventing the wheel, as they won't even be able to formulate a web search for ongoing CS work related to their own.

This problem remains unsolved. Certainly people like Mateas, who teach procedural literacy courses from an extensive background in computer science, no doubt help their students make such connections. But for our health as a field we need to begin to make appropriate connections with CS in a less ad-hoc way. Simply recognizing that our discipline is closely connected to CS, and that

it would be valuable for our students to learn some CS concepts and vocabulary, is doubtless the first step.

Personally, part of my motivation for editing *The New Media Reader* (Wardrip-Fruin and Montfort) was to present CS research papers alongside other fundamental documents for our history as a discipline. And I'm glad to report a last bit of anecdotal evidence from two faculty friends who recommend the book to their students—both report that, while the CS papers are often a struggle for those approaching the book from humanities and arts perspectives, they also work with students attracted to the *NMR* because of its CS content who find themselves challenged as they work through the contributions from artists and cultural critics. If our discipline is inspiring students to come to grips more fully with the approaches and concepts of other fields, we must be doing something right.

4 An Introductory Writing Course

The editors of this collection have asked that I include a note about my own teaching experiences with digital literature, which have largely operated from the creative (rather than interpretive) direction. Though I have taught such courses at New York University, Brown University, the Summer Literary Seminars, and other venues, I have recently moved institutions (to the University of California, Santa Cruz) and am not yet teaching a current course in this area. But I am happy to talk about a past course.

My most recent teaching in the field is a 10-week course titled *Writing for Digital Media* that I taught for two years in the Communication department at the University of California, San Diego. This course presented the particular challenge that the students involved were, with a few notable exceptions, having their first coursework in literary writing of any kind, completely unexposed to computer science concepts (or even the structured text editing used in wikis and similar environments), and without background other than personal experience in the history or present of digital media. Luckily, given the interdisciplinary nature of the department, my colleagues were happy for me to structure the course as a themed introduction to all three areas, emphasizing their connections. So the range of activities in our course meetings included in-class experimental writing exercises, seminar-style discussions of background readings and digital works, lab-style computer tutorials and debugging assistance, and reading/viewing/playing student projects and offering workshop-style critique. While many of these activities were new to the students, registration priority meant that most enrolled students were in their third or final year of the Communication degree program, and already had extensive experience think-

ing about media forms, representations, institutions, and production processes—an invaluable foundation for our discussions.

The primary emphasis of the course projects was helping students develop a sense of the relationship between writing and procedural systems. Given the background of the students, and my own interests, the most effective way to accomplish this was through an emphasis on writing and games, from Oulipian language games through the structures of mainstream computer role-playing games (RPGs). Outside of course meetings (and preparation for them) student work was organized by five milestones:

1. *A sticker writing project.* This project didn't involve creating a computational system along with writing, but was designed to get students thinking about literary writing in new ways (and practicing the short, situated chunks of writing common to working in many digital forms). It was inspired by projects such as Nick Montfort and Scott Rettberg's *Implementation* and Robert Kendall's *Logozoa,* but with a greater connection to particular locations. Students wrote at least five original texts specific to a place, used stickers to post the texts in that place, took photos documenting the placement (at least one showing context and one with legible text, for each sticker), and then created a blog post about the project. Within this loose format students created works with many structures, ranging from memoir via placement on personally-meaningful objects to scavenger-hunt style directions and low-tech locative narratives.

2. *A story-game project.* The next project required thinking about writing and systems, but employed media for system creation more easily authored and revised (for most students) than digital computation: tabletop board games, card games, and other types of traditional games. These are also media in which many students had experience thinking about structures of rules, because human players are required to uphold the rules of traditional games, and reason through any ambiguities in the rules, whereas computer games hide many of the operations of their rules. Inspired by projects such as *Once Upon a Time* (by Richard Lambert, Andrew Rilstone and James Wallis) and *Betrayal at the House on the Hill* (by Bill McQuillan, Bruce Glassco, Mike Selinker, Rob Daviau, and Teeuwynn Woodruff), students were encouraged to consider both game mechanics that could elicit storytelling from players and those that produced a version of a pre-written narrative through play—though all projects had to include a significant amount of original writing. While, unsurprisingly, the most common project concept was to produce altered versions of familiar boardgames that told stories of college life, the designs and texts were varied and often engaging. Further, during both the design process and

project critique, students began active discussion of how rule systems express ideas both through their structures and the play experiences produced.

3. *A tool tutorial.* The computer role-playing game *Neverwinter Nights* was designed to support player-produced "modules" of content. The included Aurora toolset allows relatively non-technical authors to define spaces, craft character dialogue trees, define quest stages and objectives, place encounters, and so on. A significant online community shares modules they have created, additional objects for use in authoring, and tips on both basic and advanced topics. The game and toolset run on modestly-powered Windows machines (and on emulators) and are available cheaply. This assignment required students to complete an Aurora toolset tutorial, providing a route to (and deadline for) basic competence on the part of non-technical students, while also showing those with more computational experience the route toward extending the tool's behavior through NWScript. For students familiar with computer role-playing games (nearly all of them, given that one course assignment was to play part of the main quest in *Neverwinter Nights*) it also gave insight into some of the common computational structures used to support games of this type, resulting in discussions of how these structures shape authoring and exprience.

4. *An RPG module.* For this project students created at least three speaking characters and two connected spaces in Aurora, using them to structure the experience of a multi-stage story (noted as updates in the reader/player's quest journal), with elements of the story (e.g., characters, locations, events, items) distributed in space, coming to some form (or multiple forms) of conclusion. Students used the largely fantasy-themed elements available in Aurora for purposes ranging from new takes on fairy tales to modern fictions in unusual settings (e.g., a costume party or a hedge labyrinth). Some gave their reader/players traditionally heroic tasks to complete, while others offered more prosaic or ambiguous undertakings, and a few forced genuinely uncomfortable choices on any who wished to reach a conclusion. Simply completing the assignment required a certain amount of procedural thinking combined with crafting language (e.g., ensuring that a character spoke appropriate lines over multiple conversations as the story progressed through stages).

5. *A final project.* Beginning either with their story-game or their RPG module, the course's final project required students to complete a significantly improved version of one. The approach combined that traditional in

writing courses (responding to workshop feedback) with that traditional in game design (playtest, revise, playtest, revise). Many students deepened their engagement with both the writing and the system for their project, and it was with this milestone that the course moved from looking like a collection of assignments to a set of projects that mattered to their authors.

While the course discussed here is rather different from that assumed in the earlier sections of this text—especially in its emphasis on writing computational fictions, rather than interpreting digital literature—I hope these notes are useful. Perhaps one or more of these assignments might be worth adding to a digital literature course, given that experience with creation can offer new insights during interpretation (as I saw when students discussed computer RPGs after using Aurora). Alternately, seeing in some detail what I advocate for training digital literature's creators may shed some light on the reasons I advocate the approach above for training digital literature's interpreters. My hope is that these two aspects of the field will move together in tandem toward a richer engagement with all the elements of digital literature.

Notes

1 An object-oriented approach breaks down a computational process into cooperating objects that usually have defined capabilities, store state information internally, and expose particular interfaces for retrieving and changing that information. Each particular object is an instance of some class (e.g., the room class) which can be subclassed (e.g., for particular types of rooms, like hallways). This is different, for example, from designing in a function-oriented way, which tends to store state information more centrally and is often approached from the perspective of what will happen rather than what the components are. This is sometimes discussed as a difference between a focus on "nouns" and "verbs."

2 The Z-machine is a "virtual machine" for running computer programs, much like the Java virtual machines that allow the same Java program to run on multiple computers (any computer for which there is an implementation of the Java virtual machine). The Z-machine was created by Infocom, and was the machine targeted by their interactive fictions. Its extremely limited capabilities (by today's standards) allowed it to be implemented for 1980s machines from Kaypro, Osborne, Atari, and many others (I've played Infocom games on all three of these). It now runs happily on the lowliest handheld computers, as well as the most powerful

workstations. But its limitations are seen as a bane by some interactive fiction authors, some of whom have begun to target their creations to other platforms.

Works Cited

Black, Maurice J. "The Art of Code." Diss. U of Pennsylvania, 2002.

Greenberger, Martin, ed. *Management and the Computer of the Future.* Cambridge, MA: MIT P, 1962.

Guzdial, Mark. "A media computation course for non-majors." *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education.* ACM Press, 2003. 104-08.

Kendall, Robert. *Logozoa.* 2006-2009. 21 Aug. 2009 <http://www.logozoa.com/>.

Lambert, Richard, Andrew Rilstone, and James Wallis. *Once Upon a Time: The Storytelling Card Game.* St. Paul: Atlas, 1996.

Mateas, Michael. "Procedural Literacy: Educating the New Media Practitioner." *On The Horizon.* 13.1 (2005).

Nelson, Theodor Holm. *Computer Lib/Dream Machines.* Self published, 1974.

McQuillan, Bill et al. *Betrayal at the House on the Hill.* Renton: Avalon, 2004.

Rettberg, Scott, and Nick Montfort. *Implementation.* 2005. 21 Aug. 2009 <http://www.nickm.com/implementation/>.

Squire, Kurt D. "Educating the Fighter: Buttonmashing, Seeing, Being." *On The Horizon* 13.2 (2005).

Wardrip-Fruin, Noah, and Nick Montfort, eds. *The New Media Reader.* Cambridge, MA: MIT P, 2003.